ECE 385

Spring 2021

Experiment 7

SOC with AES Encryption/Decryption

Wenhao Tan & Jiacheng Huang
ABH
Jiaxuan Liu

Introduction
In this lab, we have implemented the 128-bit Advanced Encryption Standard in Quartus as an IP (intellectual property) core for both encryption (software IP core) and decryption (hardware IP core). The main idea behind AES encryption is to use the given plain text key and message, go through a series of functions that modifies the sequence or data of the message, and result in an encrypted message. To decrypt, we use the same key and by reversing the functions, we can get back the original plain text message before encryption. We can then use the benchmark to see that hardware decryption is much faster than software encryption.

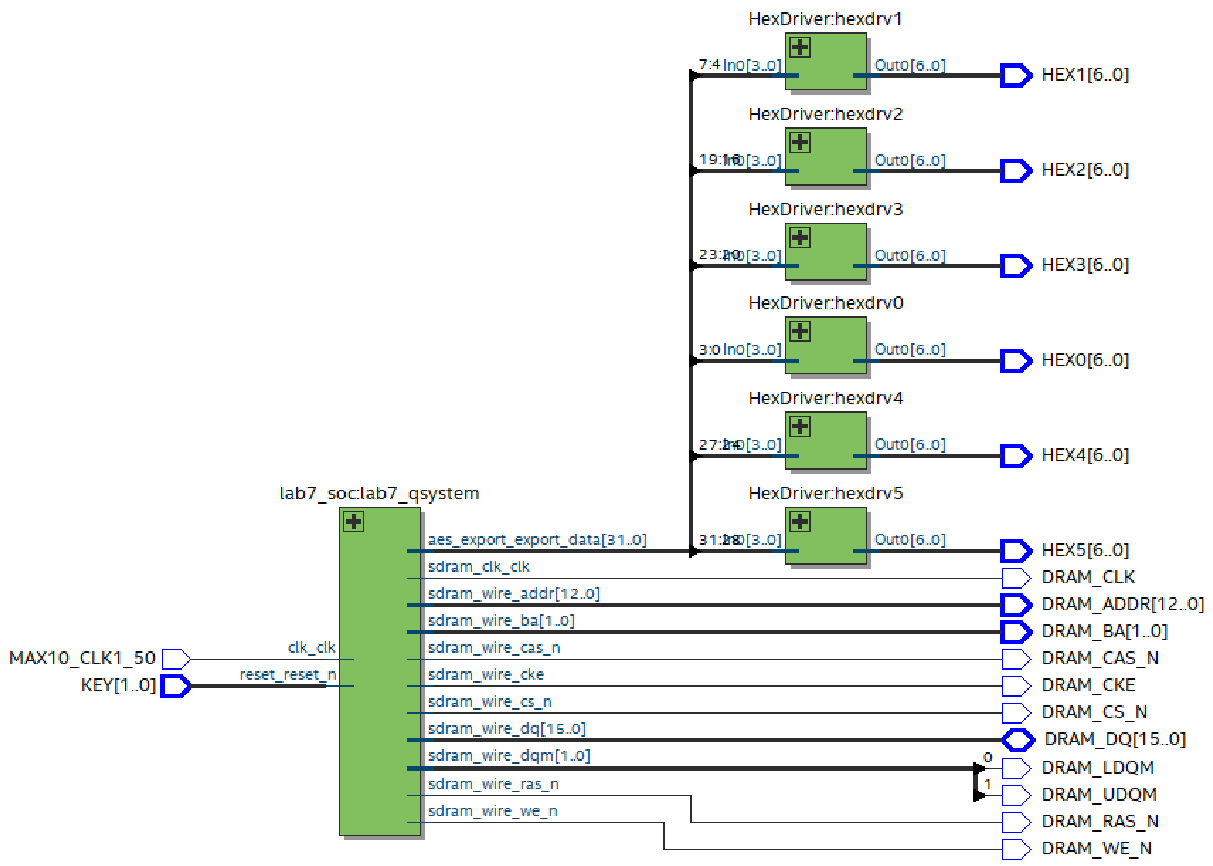
Written Description of Software Encryptor
The encryption process is made up with five subroutines: KeyExpansion, AddRoundKey, SubBytes, ShiftRows, and MixColumns, making use of the S-Box, gf_mul, and Rcon tables, which are hardcoded into a C header file AES.h.
During the KeyExpansion subroutine, the input key is expanded into a total of 11 round keys (through some S-Box transformation and Rcon XOR).
In the SubBytes subroutine, each of the 16 states is transformed by changing each state value with a corresponding value in the S-Box table.
In the ShiftRows subroutine, each row of states is right shifted by the number of states equal to the index of the row, starting with index 0.
In the AddRoundKey subroutine, the round keys are used to do XOR operations with the states in order to update the states.
In the MixColumns subroutine, the four words (or the columns of states) are linearly transformed using the GF2^8 algorithm and the gf_mul table to produce a new word.

Written Description of Hardware Decryptor
The decryption is controlled by a finite state machine we wrote in AES.sv. The FSM simply controls which function should be used next in the decryption process. There are 4 main functions in the decryption, InvShiftRow, InvSubByte, AddRoundKey, and InvMixCol. We first have to generate 11 round keys using the KeyExpansion functions. Then, we decrypt starting from the last round key, looping through (InvShiftRow, InvSubByte, AddRoundKey, InvMixCol) 9 times, and then InvShiftRow, InvSubByte, AddRoundKey one more time before decryption is complete. The decryption process starts when AES_START goes high, and when decryption is finished, we set AES_DONE to high.

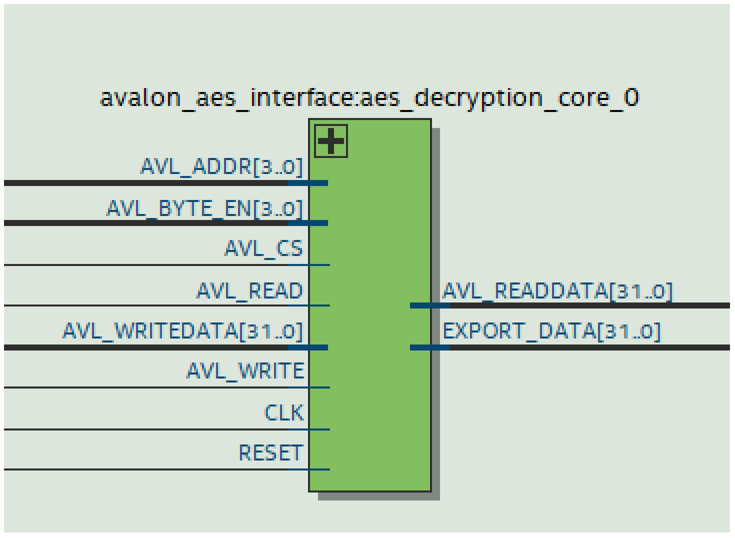
Written Description of avalon_aes_interface.sv
The register file is created by allocating a 16 by 32bits array block. The first four 32bits stores the AES key; the next four 32bits stores the AES encrypted message; the next four 32bits stores the AES decrypted message; the next two 32bits are unused; the next 32bits stores the start signal of the decryption process, however, only the last bit is used; the final 32bits stores the done signal of the decryption process, also only the last bit is usd.

We can access the register file in NIOS II by using a pointer to the starting address of the register file. Then we can modify the AES key, encrypt/decrypt messages, and start/done signals.

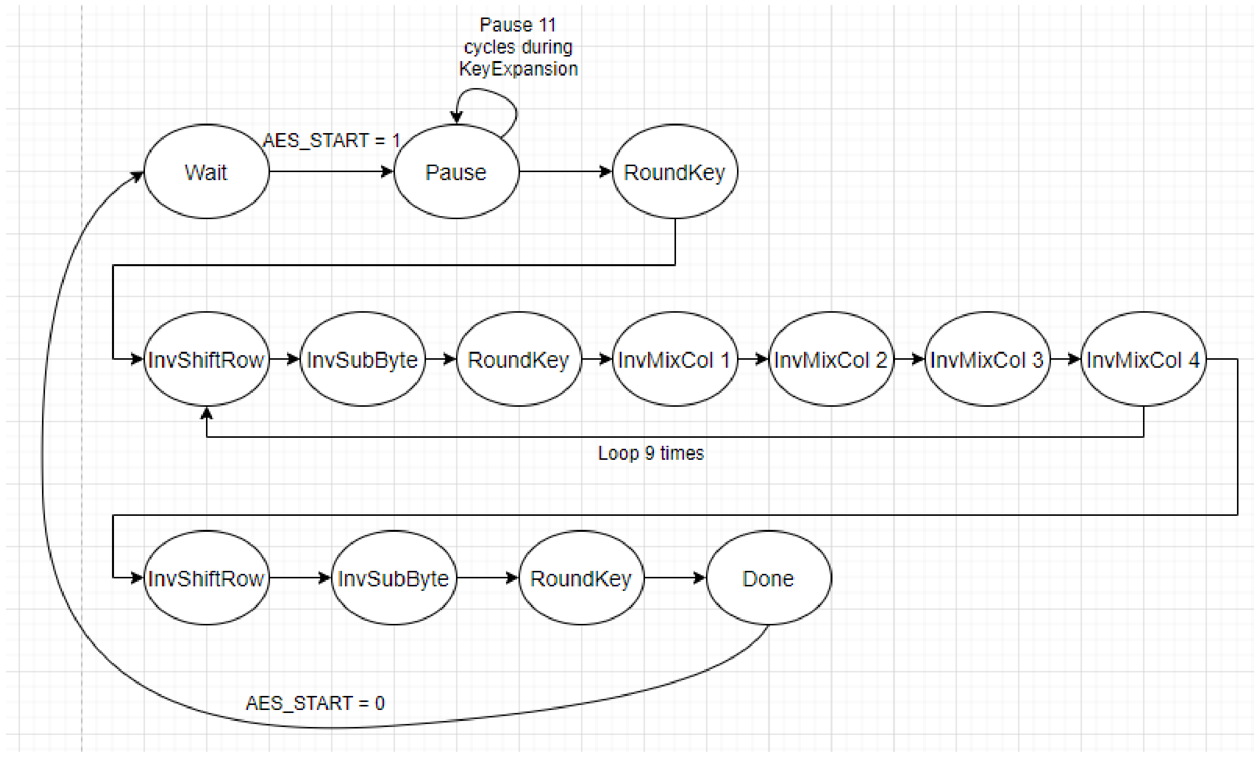
Top Level RTL Diagram



RTL View of avalon_aes_interface.sv



State Diagram of AES Decryptor Controller



Module Descriptions

Module: `lab7.sv`

Inputs: `MAX10_CLK1_50, [1:0]KEY`

Outputs: `[6:0]HEX0-5, SDRAM logics`

Description: We instantiate the `lab7_soc` module inside to configure the SDRAM connections, as well as exporting the first 2 bytes and last byte of the AES key. These 3 bytes will then be displayed on the FPGA Hex displays.

Purpose: This is the top level module for lab 7. This module is used to set up all the connections between the FPGA and SDRAM and the communication between the FPGA and NIOS II.

Module: `KeyExpansion.sv`

Inputs: `clk, [127:0] Cipherkey`

Outputs: `[1407:0] KeySchedule`

Description: We use the input `Cipherkey` to generate 10 more round keys, each one is modified from the previous one through some SubByte operation and XOR with the Rcon table.

Purpose: This module is used to generate 10 new round keys from the given input key, then outputting the combined 11 round keys.

Module: `SubBytes.sv`

Inputs: `clk, [7:0] in`

Outputs: `[7:0] out`
Description: We use the S-Box table, which is hardcoded into the module, to change the input byte to a different corresponding value in the table and output it. The `InvSubBytes.sv` is the same except using the Inverse S-Box table. We have to instantiate 16 of this modules, one for each state.
Purpose: This module is used to reverse the SubBytes subroutines done in AES encryption.

Module: `InvShiftRows.sv`
Inputs: `[127:0] data_in`
Outputs: `[127:0] data_out`
Description: We update the states by hard coding the row shifting.
Purpose: This module is used to reverse the ShiftRows subroutines done in AES encryption.

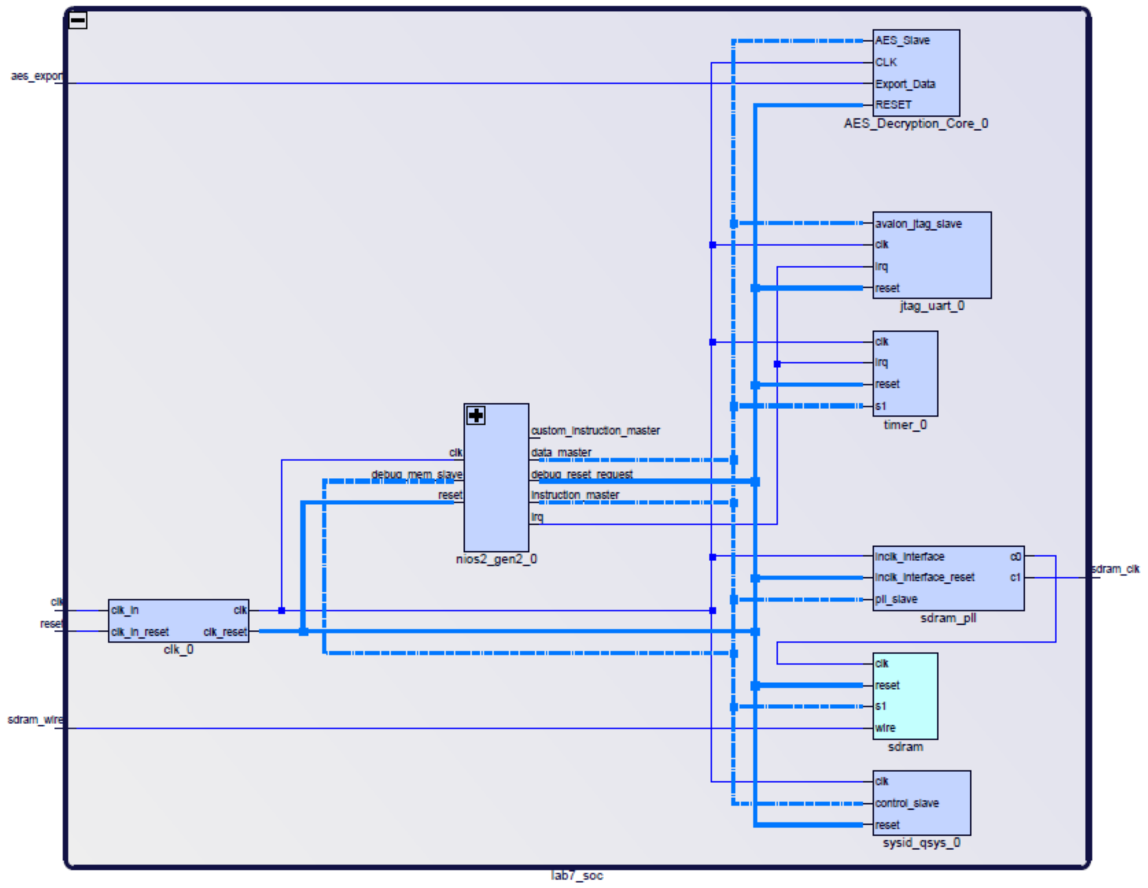
Module: `InvMixColumns.sv`
Inputs: `[31:0] in`
Outputs: `[31:0] out`
Description: We update the columns of states by using the GF2^8 algorithm and performing finite field multiplications with 0x9, 0xb, 0xd, 0xe. We have to run this module 4 times to complete one InvMixColumn subroutine because it can only reverse one column at a time.
Purpose: This module is used to reverse the MixColumns subroutines done in AES encryption.

Module: `AddRoundKey.sv`
Inputs: `[127:0] in, key`
Outputs: `[127:0] out`
Description: We XOR the input with the key and output it.
Purpose: This module is the same as the AddRoundKey subroutine in AES encryption.

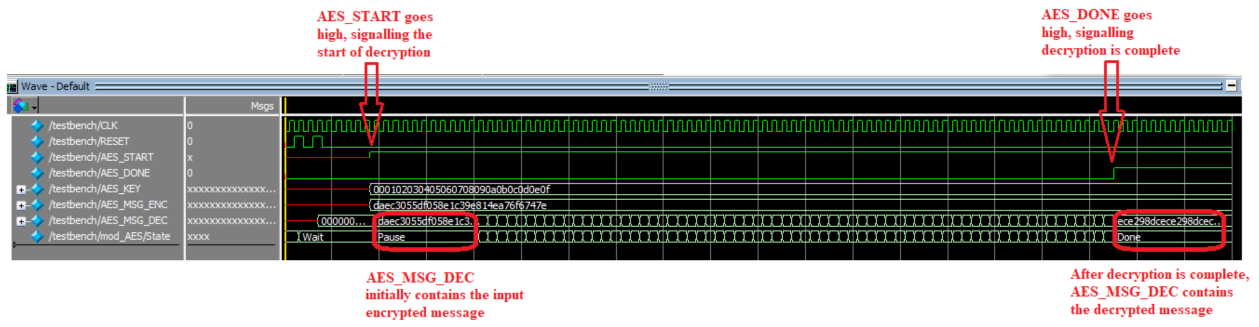
Module: `AES.sv`
Inputs: `CLK, RESET, AES_START, [127:0] AES_KEY, [127:0] AES_MSG_ENC`
Outputs: `AES_DONE, [127:0] AES_MSG_DEC`
Description: We use the input AES key and encrypted messages and start the decryption process when the AES_START signal goes high. We generate the 11 round keys using the KeyExpansion module. The generate key takes 11 clock cycles, so we have to pause for that amount of time before starting the next subroutines. Then performing a series of InvSubBytes, InvShiftRows, AddRoundKey, InvMixColumns subroutines following the finite state machine, and outputting the AES decrypt message after decryption is done and AES_DONE goes high.
Purpose: This module is the complete hardware decryption process and we instantiate all the function modules in here.

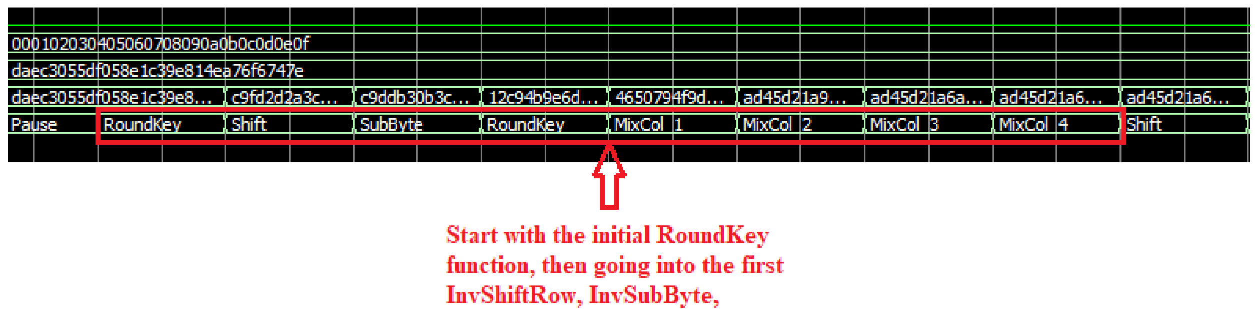
Module: `avalon_aes_interface.sv`
Inputs: `CLK, RESET, AVL_READ, AVL_WRITE, AVL_CS, [3:0] AVL_BYTE_EN, [3:0] AVL_ADDR, [31:0] AVL_WRITEDATA`
Outputs: `[31:0] AVL_READDATA`
Description: We create a 16 by 32 bits register file in this module. We also need to store the AES decrypted message in the register file with array index 8-11, and get the AES key from array index 0-3, the AES encrypted message from array index 4-7. The AES_START signal is stored in array index 14, and AES_DONE signal is stored in array index 15. When the `AVL_WRITE` signal is high, we store the `AVL_WRITEDATA` data into the register file with array index `AVL_ADDR`, but we only store the bits that are high in `AVL_BYTE_EN`. When the `AVL_READ` signal is high, we read from the register file with array index `AVL_ADDR` into `AVL_READDATA`. We also store the first 2 bytes and last byte of the AES key in 32 bits `EXPORT_DATA` which will be used to display on the FPGA Hex displays.
Purpose: This module is used to communicate the result of the hardware decryption with the NIOS II. We can read and write from the NIOS II software using the address of the register file in the module.

System Level Block Diagram



Annotated Simulation of the AES Decryptor


AES_START goes high, signalling the start of decryption
AES_DONE goes high, signalling decryption is complete
Wave - Default
Msgs
/testbench/CLK 0
/testbench/RESET 0
/testbench/AES_START x
/testbench/AES_DONE 0
/testbench/AES_KEY xxxxxxxxxxxxxx...
/testbench/AES_MSG_ENC xxxxxxxxxxxxxx...
/testbench/AES_MSG_DEC xxxxxxxxxxxxxx...
/testbench/mod_AES/State xxxx
000102030405060708090a0b0c0d0e0f
daec3055df058e1c39e814ea76f6747e
000000...
daec3055df058e1c3...
ece298dcece298dcec...
Wait
Pause
Done
AES_MSG_DEC initially contains the input encrypted message
After decryption is complete, AES_MSG_DEC contains the decrypted message



Answers to Post-lab Questions

- Which would you expect to be faster to complete encryption/decryption, the software or hardware? Is this what your results show?

We would expect the hardware decryption to be much faster than the software encryption. This is also what we saw in the results of the benchmark. Hardware can reach 200KB/s, while software is only less than 0.5KB/s.

- If you wanted to speed up the hardware, what would you do? (Note: restrictions of this lab do not apply to answer this question)

To speed up the hardware, we can instantiate more InvMixColumn modules so that part of the operation is faster. We can also make all the modules 128bits and use pipelining so the idle modules can have something to do.

Design Resources and Statistics

LUT	6378
DSP	0
Memory (BRAM)	562176
Flip-Flop	2812

Frequency	111.93 MHz
Static Power	96.18 mW
Dynamic Power	0.68 mW
Total Power	105.87 mW

Problems

One of the problems we ran into during the lab was that we did not fully understand where we should instantiate the AES module. We initially instantiated it in our top level lab7.sv, but we ran into the problem that we do not have access to the input logics. The input logics are not exported out of avalon_aes_interface.sv, so we removed the instantiation in lab7.sv and instantiated it in avalon_aes_interface.sv instead.

The other problem that we had was that our simulation was functioning correctly, but the console output of the NIOS II is off by 8 bits. We fixed the error by changing AVL_READDATA to always_comb from always_ff. This is because we want to read data instantly and not during the next rising edge of clk.

Conclusions

In conclusion, we have learned the AES encryption and decryption methods both through software and hardware implementations. We wrote the KeyExpansion, AddRoundKey, SubBytes, ShiftRows, and MixColumns subroutines in C, and the reverse modules in Systemverilog. During this process, we learned that hardware encryption/decryption is way faster than software.